National University of Computer and Emerging Sciences

Design Analysis and Algorithms Project Report

STUDENTS NAME:	Sheir Muhammad, Manahil Shakeel, Ahmed Baig
GROUP NUMBER:	L
REGISTRATION NUMBER:	20I-0958, 20I-2302, 20I-1884
DEGREE PROGRAM:	BSSE
SECTION:	R
SUBJECT NAME:	Design Analysis and Algorithms
DATE OF SUBMISSION:	December 4th, 2022
SUBMITTED TO:	Mr. Bilal Khalid Dar

Machine Specifications:

Member 1

>Device name	HP Pavilion 15 Notebook PC
Processor	Intel (R) Core (TM) i3-4030U CPU 1.90GHz
Installed RAM	4GB
Device ID	35434434-3239-3231-4348-6CC217712159
Product ID	00330-80000-00000-AA294
System type	64- bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Member 2

>Device name	DESKTOP-GF1H03L
Processor	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 1.69 GHz
Installed RAM	8.00 GB (7.73 GB usable)
Device ID	3DF78ACB-BD7C-4913-9913-913C5E21EF37
Product ID	00330-80000-00000-AA294
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Member 3

>Device name	LAP-727
Processor	Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
Installed RAM	8.00 GB (7.84 GB usable)
Device ID	6C2524F5-F89A-4242-A869-AFAEF6C15EB8
Product ID	00330-53029-53059-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Details Of Data Set:

soc-sign-bitcoin-alpha	Weighted, Signed, Directed, Temporal	Nodes: 3,783	Edges: 24,186	Bitcoin Alpha web of trust network

Complexity Analysis:

>For Prims Algorithm

- If we consider the above method, both the average case and the worst-case time complexities are the same as stated above. Which is, O(V(log(v)) + E(log(V))).

The best case time complexity for decrease Key operation is O(1) which makes the best case complexity as O(E(log(V))).

>For Kruskal's Algorithm

- O(E(log(E) + E^2 + V)). Including only the high ordered terms, the runtime becomes O(E(log(E) + E^2)).

>For Average Degree

- **Time Complexity:** O(V+E)

>For Breadth-First Search Algorithm

- **Time Complexity:** O(V+E), where V is the number of nodes and E is the number of edges.
- **Space Complexity**: O(V)
- Follow the below method to implement BFS traversal.
  -Declare a queue and insert the starting vertex.
  -Initialize a visited array and mark the starting vertex as visited.
  -Follow the below process till the queue becomes empty:
  -Remove the first vertex of the queue.
  -Mark that vertex as visited.
  -Insert all the unvisited neighbors of the vertex into the queue.

>For Depth-First Search Algorithm

- **Time Complexity:** O(V+E), where V is the number of nodes and E is the number of edges in the graph.
- **Space Complexity**: O(V) since an extra visited array of
  size V is required.
- Follow the below method to implement DFS traversal.
  -Create a recursive function that takes the index of the node and a visited array.
  -Mark the current node as visited and print the node.
  -Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

>For Cycle Detection in a Graph Algorithm

- **Time Complexity:** O(V+E), the Time Complexity of this method is the same as the time complexity of DFS traversal which is O(V+E).

- **Space Complexity**: O(V). To store the visited and recursion stack O(V) space is needed.
- Follow the below method to implement Cycle Detection of a graph.
  - Create the graph using the given number of edges and vertices.
  -Create a recursive function that initializes the current vertex, visited array, and recursion stack.
  -Mark the current node as visited and also mark the index in the recursion stack.
  -Find all the vertices which are not visited and are adjacent to the current node and recursively call the function for those vertices
  -If the recursive function returns true, return true.
  -If the adjacent vertices are already marked in the recursion stack then return true.
  -Create a wrapper function, that calls the recursive function for all the vertices, and
  -If any function returns true return true.
  -Else if for all vertices the function returns false return false.

>For Bellman-Ford Algorithm

- **Time Complexity:** The runtime complexity of the algorithm is O(v*e).
- **Space Complexity**: O(V).
- The working of the Bellman-Ford algorithm is the same as Dijkstra's algorithm. The only difference is that it does not use the priority queue. It repetitively loops over all the edges and updates the distances at the start node, the same as in Dijkstra's algorithm.
  If a graph G=(V, E) contains a negative weight cycle, then some shortest paths may not exist. Bellman-Ford algorithm finds all shortest path lengths from a source s ϵ V to all v ϵ V or determines that a negative weight cycle exists.
  Given a weighted directed graph G(V, E) with source (s) and weight function w: E -> R, the algorithm returns a boolean value TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source. The algorithm produces the shortest path and its weights. If there is such a cycle, the algorithm indicates that no solution exists.

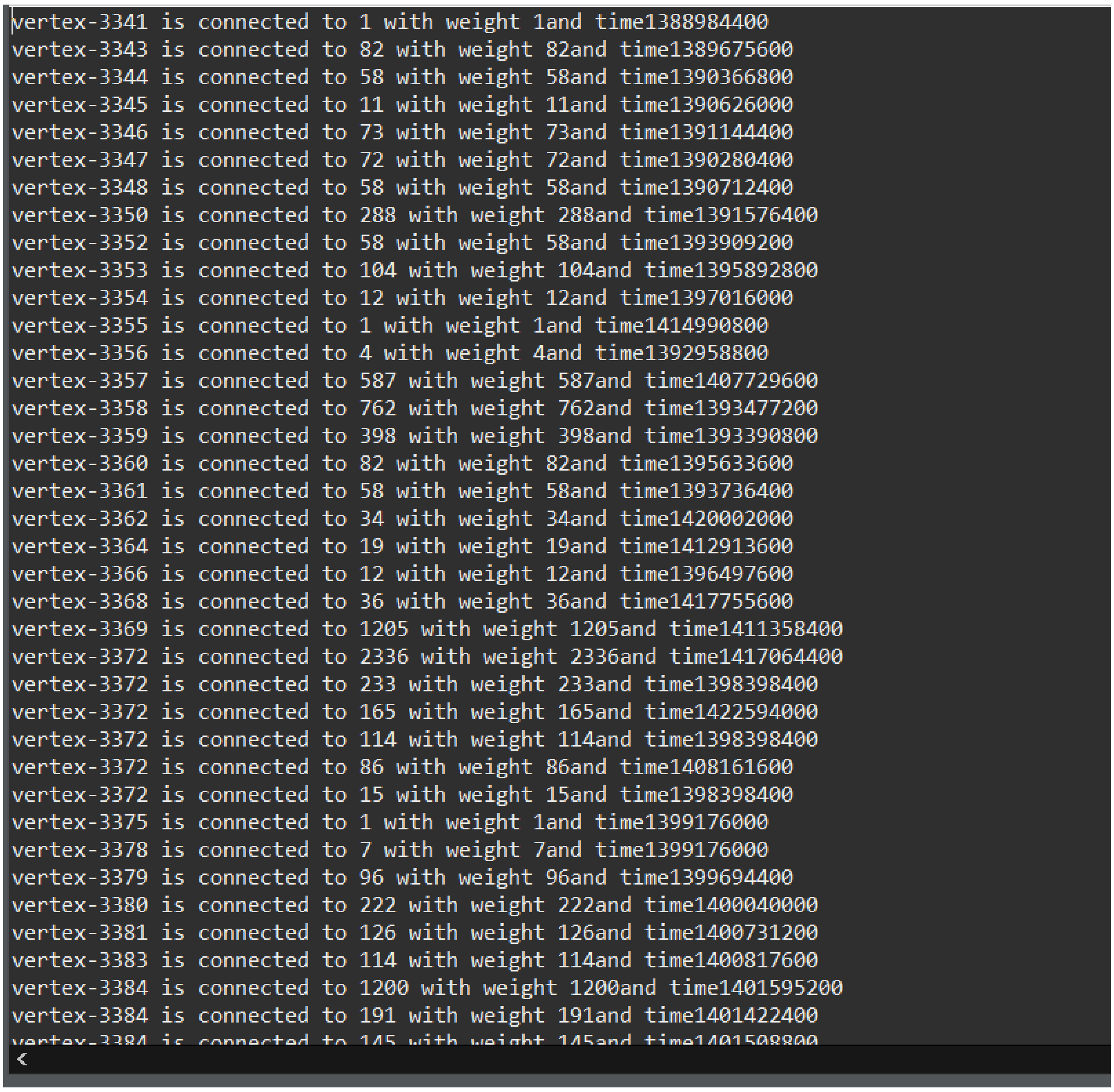
>For Dijkstra’s Algorithm

- **Time Complexity:** O(V + E logV).
- **Space Complexity**: O(V+ E)

>For Diameter of Graph

- **Time Complexity:** O(V * E).

- **Sample Graph:**

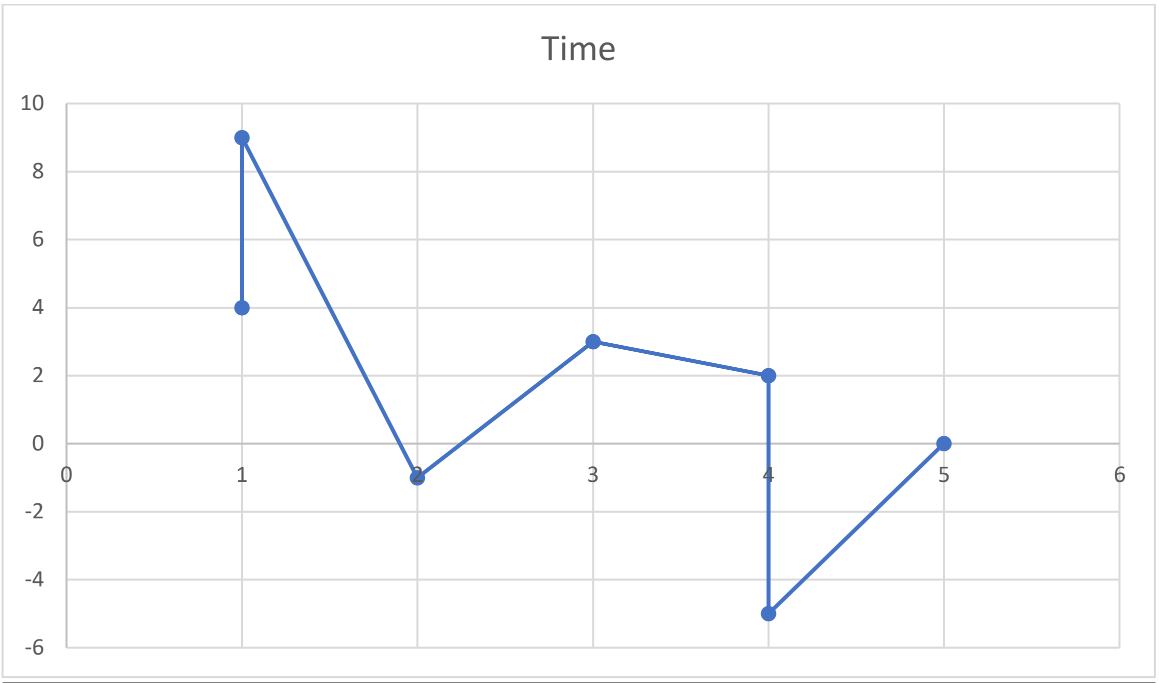


```
vertex-3341 is connected to 1 with weight 1and time1388984400
vertex-3343 is connected to 82 with weight 82and time1389675600
vertex-3344 is connected to 58 with weight 58and time1390366800
vertex-3345 is connected to 11 with weight 11and time1390626000
vertex-3346 is connected to 73 with weight 73and time1391144400
vertex-3347 is connected to 72 with weight 72and time1390280400
vertex-3348 is connected to 58 with weight 58and time1390712400
vertex-3350 is connected to 288 with weight 288and time1391576400
vertex-3352 is connected to 58 with weight 58and time1393909200
vertex-3353 is connected to 104 with weight 104and time1395892800
vertex-3354 is connected to 12 with weight 12and time1397016000
vertex-3355 is connected to 1 with weight 1and time1414990800
vertex-3356 is connected to 4 with weight 4and time1392958800
vertex-3357 is connected to 587 with weight 587and time1407729600
vertex-3358 is connected to 762 with weight 762and time1393477200
vertex-3359 is connected to 398 with weight 398and time1393390800
vertex-3360 is connected to 82 with weight 82and time1395633600
vertex-3361 is connected to 58 with weight 58and time1393736400
vertex-3362 is connected to 34 with weight 34and time1420002000
vertex-3364 is connected to 19 with weight 19and time1412913600
vertex-3366 is connected to 12 with weight 12and time1396497600
vertex-3368 is connected to 36 with weight 36and time1417755600
vertex-3369 is connected to 1205 with weight 1205and time1411358400
vertex-3372 is connected to 2336 with weight 2336and time1417064400
vertex-3372 is connected to 233 with weight 233and time1398398400
vertex-3372 is connected to 165 with weight 165and time1422594000
vertex-3372 is connected to 114 with weight 114and time1398398400
vertex-3372 is connected to 86 with weight 86and time1408161600
vertex-3372 is connected to 15 with weight 15and time1398398400
vertex-3375 is connected to 1 with weight 1and time1399176000
vertex-3378 is connected to 7 with weight 7and time1399176000
vertex-3379 is connected to 96 with weight 96and time1399694400
vertex-3380 is connected to 222 with weight 222and time1400040000
vertex-3381 is connected to 126 with weight 126and time1400731200
vertex-3383 is connected to 114 with weight 114and time1400817600
vertex-3384 is connected to 1200 with weight 1200and time1401595200
vertex-3384 is connected to 191 with weight 191and time1401422400
vertex-3384 is connected to 145 with weight 145and time1401508800
```

Table and Performance Graph:

V	D	T
1	2	4
1	4	9
2	3	-1
3	6	3
4	3	2
4	5	-5

5	6	0


Time